JAN AXELSON

USB Debug Tips

USB is a flexible, high-speed replacement for serial and parallel ports. But flexible also means complicated —it's much harder to debug a USB design and qualify your product's compliance.

The universal serial bus (USB) is a popular interface for devices that must communicate with personal computers. For the past several years, all new PCs and Macs have included USB support. The interface is flexible enough to use for common peripheral types like drives and keyboards, as well as custom, application-specific designs. And USB has features that appeal to both end users and developers, including the ability to power devices from the bus, easy bus expansion with hubs, and automatic identification of devices by the host computer.

But USB's capabilities mean that the interface is more complex than the older serial and parallel interfaces it replaces. Every USB device must respond to a set of standard requests and other events on the bus. Most bus transactions require two-way communications that must complete with minimal delays. And the data on the bus is encoded and thus not easily viewed on an oscilloscope or logic analyzer alone.

You can simplify and speed up the development and troubleshooting of USB devices with the tools and techniques you choose and the design decisions you make. This article presents the options and how to take advantage of them.

USB basics

The USB specification is the product of Intel, Microsoft, and several other companies involved with PCs and peripherals. The specification document and related information and tools for developers are available at the website of the USB Implementers Forum, known as USB-IF for short (*www.usb.org*).

Every bus has a host controller that controls communications with devices on its bus. To increase the bandwidth available for devices, a single computer can have multiple host controllers, each controlling its own bus.

USB supports three bus speeds: low speed at 1.5Mbps, full speed at 12Mbps, and high speed at 480Mbps. High speed was added in version 2.0 of the specification, which was published in 2000. Windows XP is the first Windows edition to support USB 2.0. Microsoft has promised USB 2.0 updates for Windows 2000 and Windows ME. Other operating systems have 2.0 support in progress as well.

For embedded PCs, Windows CE also supports USB. Most Windows CE computers function as USB hosts, but Windows CE 3.0 also includes drivers for Cypress/ScanLogic's SL11 host/slave controller. Using these drivers, or similar drivers for other controllers, a Windows CE computer can function as a USB peripheral.

Much of USB's versatility is due to its four transfer types, each suited for different purposes. *Control transfers* carry requests used in the enumeration process and are also available for other communications that send requests to a device and (optionally) receive a reply. *Interrupt transfers* are for devices such as keyboards and mice, where the host periodically requests or sends data. *Bulk transfers* are for devices such as printers and scanners, where fast transfers are desired, but the data can wait if the bus is busy. *Isochronous transfers* are for real-time audio and other applications where timing is critical and occasional errors can be tolerated.

On boot-up or when a device is attached to a bus, the device's hub reports the attachment to the host computer. In a process called enumeration, the host sends a series of requests

You can simplify and speed up the development and troubleshooting of USB devices with the tools and techniques you choose and the design decisions you make.

to learn about the device and establish communications with it. The device returns information in data structures called descriptors. Windows' Device Manager compares the information in the descriptors with the information in the PC's INF files. Device Manager finds the best match and assigns a device driver that enables applications to access the device.

All devices must comply with USB's requirements for power management. These include limiting the amount of bus current the device draws and detecting when to enter the low-power Suspend state. The amount of current allowed depends on information in the device's descriptors.

Dozens of USB-capable device controller chips are available. Some are microcontrollers with a USB port. Others are CPU-less controllers with a USB port and a serial or parallel interface for communicating with a generic microcontroller. Most USB-capable microcontrollers have a C compiler available. If you have experience with a particular microcontroller family, it makes sense to see if a USB-capable variant is available.

Testing a device's USB communications comprises several stages. The first goal is successful enumeration. If the device doesn't enumerate, the interface can do little else. Other tests include using the device to carry out its intended purpose (for example, reading and writing files to a USB drive) and verifying that the device obeys the rules for power management.

Debugging of USB communications can take place in three locations: at the host PC, at the device, and in the cable. Each has its advantages.

For more about designing USB devices, see Jack Ganssle's "An Introduction to USB Development" (March 2000, p. 79) and my article, "HIDs Up" (October 2000, p. 61).

Debugging at the host

From the host, you can verify that a device has enumerated and can perform its intended functions. After detecting a problem at the host, finding the source of the problem often requires examining the device firmware or the bus traffic in the cable.

After a device attaches to a host, Windows' Device Manager (Figure 1) provides a quick check of whether the device enumerated without problems. An exclamation point over the listing's icon means that there was a problem communicating with the device or finding a driver. An X over the icon means the device is present but has been disabled.

To see exactly what information the host received during enumeration, use the USBCheck suite of applications or the new USB Command Verifier tool. Both are available free from USB-IF's website. USBCheck enables you to view descriptors, send control requests, view the results, and run further tests on devices in the hub and HID (human interface device) classes.

USBCheck's Device Framework tests read the descriptors and send standard requests. These tests are very useful as an initial check that Windows is retrieving the expected information from your device. Figure 2 shows a device descriptor that USBCheck reported receiving from a hub.

After the host has enumerated the device, applications can test the device in its intended use. A Windows device driver typically enables applications to access a device using some combination of the API functions `ReadFile()`, `WriteFile()`, and `DeviceIoControl()`. Some device classes have additional support. For example, applications can access a USB drive in the same ways they access other drives. The application doesn't have to know or care whether the drive uses USB or another interface, because these details are handled at a lower level.

For many devices, a USB class specification defines the device's expected behavior, and thus the firmware's responsibilities. Examples include HIDs, mass-storage devices, and devices for still-image capture.

When something goes wrong, the error messages returned by Windows are often of limited help. For example, when a `WriteFile()` to a HID-class device fails, a common error returned is "CRC Error." But this message can result from just about any firmware problem that caused the transfer to fail. It typically has nothing to do with an error detected in the CRC calculation for error checking. Tracking down the cause of problems like this often requires debugging at the device or on the bus.

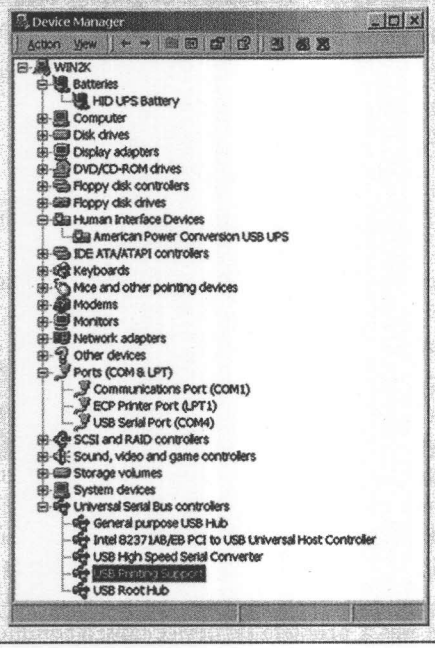
Compliance testing

The USB Implementers Forum and Microsoft offer testing opportunities for developers of USB devices and their host software. Passing the tests helps to qualify a product to display the USB logo or the Microsoft Windows logo.

For thorough testing of a product under a variety of conditions, USB-IF members can enroll a device in the Compliance Program. A one-year membership costs $2,500. The fees support the cost of running the program and other activities that support the development of USB products and the promotion of USB in the marketplace.

When a device meets the compliance program's criteria, USB-IF deems it to have "reasonable measures of acceptability" and adds it to its Integrators List of compliant devices. On receiving a signed license agreement and payment, USB-IF authorizes the device to display the USB logo.

FIGURE 1 Windows' Device Manager displays all enumerated USB devices



The compliance program's two criteria are checklists and compliance testing. The checklists contain questions relating to a product and its behavior. Separate checklists exist for vendors of peripherals, hubs, systems with USB hosts, and cables. Some products require multiple checklists.

The peripheral checklist covers mechanical design, device states and signals, and operating voltages and power consumption. Along with each question is a reference to a page in the specification where you can find more information. The checklists are available from USB-IF's website.

To help pass the electrical tests, USB-IF has a USBHSET utility with software and test procedures. Another option is VI Engineering's USB Pre-Compliance Tester, which is a hardware unit that performs all of the electrical tests in the test document. The included LabView-based software enables you to view eye patterns, rise and fall times, crossover voltages, in-rush current, and more.

To help verify proper responses in the protocols discussed in Chapter 8 of the USB specification, Professional Interactive Media Centre NV (PIMC) offers the Ch8ck utility. The tests performed by Ch8ck include sending a packet ID for an unsupported direction or transfer type, checking the response of a halted endpoint, and checking for bit stuffing when required in the CRC value.

When you can answer yes to everything on the checklists that apply to your product, you're ready for compliance testing. USB-IF sponsors workshops that enable you to test your device with different types of hardware. Every workshop has many vendors and products available. You can schedule private tests with vendors of host hardware. And you can participate in one of USB-IF's "plugfests," where as many vendors as possible connect their devices to a single host to find out if all can co-exist peacefully. USB-IF also authorizes some private labs to perform compliance tests.

The Compliance Test Procedure document has detailed descriptions of the tests, which cover responding to standard requests, power consumption and distribution, signal quality, and interoperability. Interoperability tests allow you to emulate the user's experience using your product on a system with a variety of other USB peripherals attached and in use with a variety of software.

Your device should function without ever causing a device-not-detected error or a system crash, hang, or reboot. The device must pass the tests not only on a bus with just your device, but also on a bus that connects a variety of hubs and other common peripherals.

If your device passes compliance testing, it's eligible to display the USB logo. To qualify for the logo, a high-speed device must also function at full speed. If you're not a member of USB-IF, you also must pay a logo administration fee of $1,500 every two years.

For devices that attach to PCs, Microsoft promotes Windows Hardware Quality Labs (WHQL) testing. These tests qualify devices to display a Microsoft Windows logo and to be included in Microsoft's Hardware Compatibility List. Microsoft may also include the device's driver in its Windows Driver Library.

Microsoft provides test kits for hardware and device drivers. You can download the kits that apply to your device and run the tests. When you believe your device can pass all tests, you submit a test package to an authorized testing site. The test package contains the device, any driver and related files, test logs, and fees.

You can find more information and downloads relating to WHQL at *www.microsoft.com/hwtest.*

Debugging from the device

At the device end, debugging is much the same as in any embedded system. The vendors of USB-capable microcontrollers offer development systems with monitor programs that enable setting breakpoints, single-stepping, tracing, and other tools for diagnosing problems.

The amount of firmware support required for USB communications depends on the controller chip's architecture. Good example firmware from the chip vendor or another source is also extremely helpful.

MCCI has a free USB Resource Compiler that can help in translating device-descriptor information into C data-initialization structures to be stored in a device's program memory. MCCI also offers the USB DataPump portable firmware package and an installation utility.

A low-cost alternative to specialized development kits is to use a PC as an emulated USB device for initial testing

FIGURE 2 A device descriptor retrieved from a hub

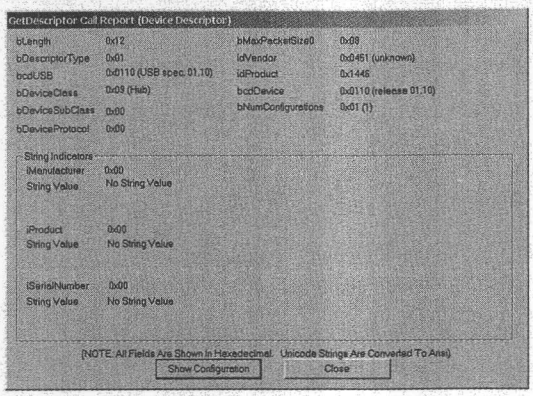
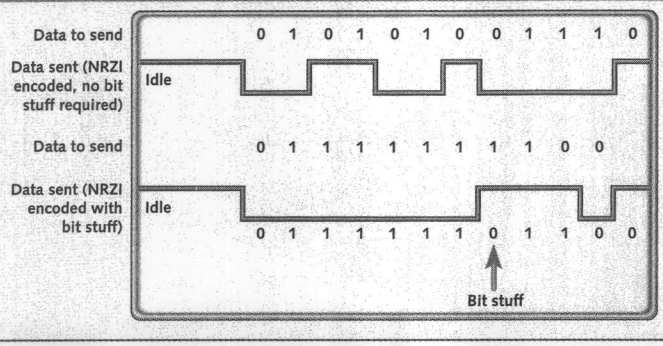


FIGURE 3 Non-Return to Zero Inverted (NRZI) encoding with bit stuffing



of code that will eventually reside in an embedded device. An example is the USBLPT-PD11 board from DeVaSys. The board contains Philips' PDIUSBD11 USB controller. The controller's I²C interface communicates with a PC's parallel port. The example software that controls the emulated device uses Borland Turbo C for DOS.

With this approach, you can write PC applications that perform the functions of the firmware that will eventually control the device, including sending descriptors during enumeration and other tasks the device is responsible for. The PC software in C will be somewhat portable to the device. Every controller has chip-specific operations, however, and will require some modification for the final product.

Debugging at the cable

Sometimes debugging at the host and device isn't enough. At the host, the information you can view and control is filtered through the host controller and its drivers. In the device, the firmware doesn't see the lowest-level communications that the hardware manages. To fill the gap, you need to view what is transmitting in the cable.

If you use an oscilloscope or logic analyzer to view USB traffic, you'll find that reading the bits isn't as easy as matching voltage levels to logic levels. Data on the bus is encoded using Non-Return to Zero Inverted (NRZI) with bit stuffing. This encoding enables the receiver to remain synchronized with the transmitter without the overhead of sending a clock signal or start and stop bits with each byte.

Instead of defining logic 0s and 1s as voltages, NRZI defines logic 0 as a voltage change, and logic 1 as a voltage that remains the same. Figure 3 shows an example. Each logic 0 results in a change from the previous state. Each logic 1 results in no change. The bits transmit least-significant-bit (LSB) first.

Bit stuffing is required because the receivers synchronize on transitions. If the data is all 0s, there are plenty of transitions. But if the data contains a long string of 1s, the lack of transitions could cause the receiver to get out of sync.

If data has six consecutive 1s, the transmitter stuffs, or inserts, a 0 (represented by a transition) after the sixth 1. This ensures at least one transition for every seven bits. The receiver detects and discards any bit that follows six consecutive 1s. The bit-stuffing overhead for random data is just 0.8%, or one stuff bit per 125 data bits.

Fortunately, the USB hardware on each end does all of the encoding and decoding, so device developers and programmers don't have to worry about it. The best way to view the data is to use a protocol analyzer that collects the data, then decodes and displays it in useful formats. You can watch what happens during enumeration, detect and examine protocol and signaling errors, view the data in any transfer, or focus on any aspect of a communication that you want.

Sources for protocol analyzers that connect to the USB cable include Catalyst, Computer Access Technology, Crescent Heart Software, Data Transit, FuturePlus, Hitex, QualityLogic, and Transdimension. Software-only analyzers that display traffic detected in the host include Perisoft's BusHound and the freeware USB Snoopy.

Any analyzer will perform the basic tasks of decoding USB traffic and displaying the results. Products differ in the user interface and in the ways they can display information. Not all analyzers support high speed. Figure 4 shows data captured using Catalyst's SBAE-20 Bus Analyzer-Exerciser. The user interface for controlling the analyzer and viewing the traffic may be a PC or a logic analyzer. An analyzer that connects to a PC may use a USB, parallel-port, Ethernet, or ISA-board connection. If you own a generic logic analyzer, a USB analyzer that connects to it may be cheaper than other options. Crescent Heart Software's analyzers connect to Tektronix's, and FuturePlus's analyzers connect to Agilent's.

Test equipment and software options

Also useful in testing and debugging is the ability to control bus traffic and signaling beyond what you can do by access-

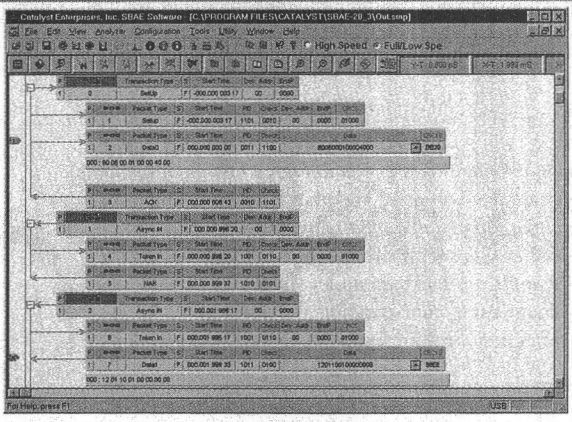
ing devices from applications. There are instruments that can do this as well.

Computer Access Technology's Traffic Generator is an example of an instrument that offers precise control over bus traffic and events. You control the Traffic Generator via a parallel-port connection to a PC running their software. You can generate both legal and illegal messages and bus conditions, and you can control the state of individual bits and the bit width.

Transdimension's USB Host/Device Exerciser and Catalyst's SBAE-20 and function both as protocol analyzers and as hosts that can generate traffic on the bus. Other useful features of the SBAE-20 include the ability to measure inrush and suspend-state currents.

RPM Systems' Root 1 USB Functional Verification Adapter performs many of the functions of a host and root hub. The Root 1 enumerates a connected device and can initiate other traffic and perform various tests, including controlling bus voltage.

FIGURE 4 Bus traffic captured with a protocol analyzer



The number and variety of testing tools has increased over the past few years as new vendors have entered the market and existing vendors have expanded and improved their products. This trend is sure to continue as USB devices gain in popularity.

That wraps up this introduction to USB debugging. As with any development project, investing in a few tools and learning to use them well can save you time and money in the long run. **esp**

Jan Axelson hosts a webpage for USB developers at www.lvr.com/usb.htm. *Portions of this article are adapted from her book,* USB Complete: Everything You Need to Develop Custom USB Peripherals, Second Edition *(Lakeview Research, 2001). Contact Jan at jan@lvr.com.*